Oberseminar Informationstechnik

Introduction

Relation to Other Courses of Our Chair

Oberseminar Informationstechnik

- VHDL Introduction
- Digital RTL Design
- Real-Valued Modelling

Neuromorphe VLSI-Systeme

- Analog CMOS circuits
- Neuromorphic systems
- EDA for analog circuits

Schaltkreis- und Systementwurf

- Design of digital systems
- Introduction to EDA software
- Simulation of digital circuits

Prozessorentwurf

- Complex digital design
- Synthesis of digital circuits
- Introduction to design flow

Overview: Structure and Contens of the Seminar

Three Main Parts of the Seminar

- **Lectures**
  - Introduction to VHDL
  - Real-Valued Modelling
  - Current Research Projects
  - Introduction to the Project Work
- **Student Presentations**
  - Topics Derived from Research
  - General Topics
    (e.g. Fuzzy Logic, Sensors, Actuators, Data Converters, etc.)
- **Project Work**
  - Digital Design + Analog Modelling
  - Functional Check through Simulation
  - Written Report

Project Work

- **ONE ACCOUNT for ALL lab courses (e.g. Schaltkreis- und Systementwurf, Prozessorentwurf, Neuromorphic VLSI, this seminar)**
- **Please REGISTER yourself for EACH LAB COURSE**

Website (Home)
https://tu-dresden.de/ing/elektrotechnik/iee/hpsn

Website (Information, Links, **Anmeldung**)
https://tu-dresden.de/ing/elektrotechnik/iee/hpsn/studium/materialien

Login via IDM (**ZIH-Login**) **required**. Typical Login ID: e.g. **s1234567**

Register for `**OSM**´

Room: **TOE 201**

Introduction to VHDL

Introduction to VHDL

- **Motivation**
  - History of VHDL
  - Fields of Application
  - Contrast to Verilog
- **Laguage Basics**
  - Entity, Architecture, Configuration
  - Data Types for Signals
  - Concurrent Processes
  - Sequential Statements
  - Hierarchical Structures

VHDL History

- **1980s: US government programme "VHSIC"**
  Very **H**igh **S**peed **I**ntegrated **C**ircuits
- **1983: VHDL defined by the VHSIC Initiative**
  VHSIC **H**ardware **D**escription **L**anguage
  - Intended as a standardized language for specification and documentation of integrated circuits and other electronics
- **1987: VHDL becomes IEEE Standard 1076**
- **1988: US DoD requires all electronic equipment to be documented in VHDL**
- **1993: Revised IEEE standard 1076**
  - Base of current VHDL
  - Newer revisions of 2002 and 2008

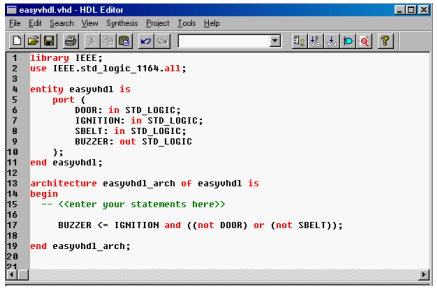
VHDL Fields of Applications

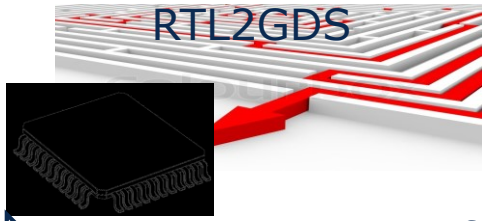
Implementation
of digital circuits

[redacted] Idea

Specification

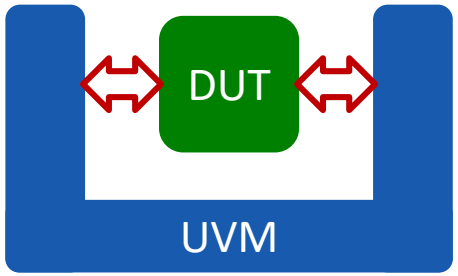
Implementation





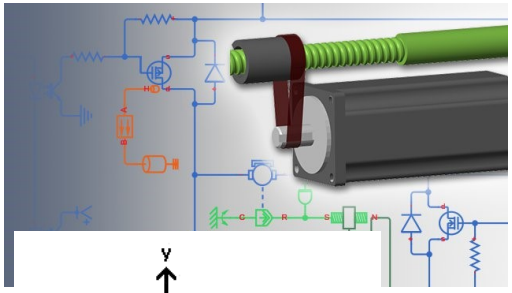
⇨ LV Prozessorentwurf

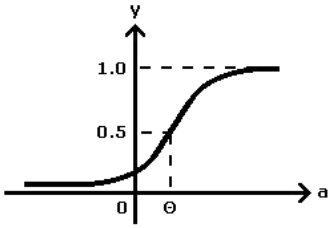
Verification
of digital and mixed-signal circuits



Simuation

Modeling
of circuits and systems






```
29
30 entity CW1_Module is
31     Port ( OE : in  STD_LOGIC;
32            LE : in  STD_LOGIC;
33            D : in  STD_LOGIC;
34            Q : out  STD_LOGIC);
35 end CW1_Module;
36
37 architecture Behavioral of CW1_Module is
38
39 begin
40 Q <= D when (OE = '0') else 'Z';
41
42 end Behavioral;
43
```

Contrast to Verilog

Compared to Verilog, VHDL …

- **provides freedom to define types**
  + good readability
  + high reusability
  – don't over-use it
- **has strong type checking**
  + can help to avoid misinterpretation of data
  – you need lots of type conversions
- **has declaration overhead**
  + very precise
  – many lines of code
- **has a set of very handy language features**
  we will come to a few of them
- **IS casE inSENsitiVE**
- **has different pitfalls**

```
architecture behave of mug is
    signal sig : std_logic_vector(7 downto 0);
begin

process (sig)
    begin
        for i in 0 to 2 loop
            sig(i) <= '0';
        end loop;
end process;

sig(3) <= '1';

end behave;
```

What is the value of sig after 10 ns?

- A: 00001000
- B: UUUU1000
- C: UUUUU000
- D: 11111000

VHDL Basics

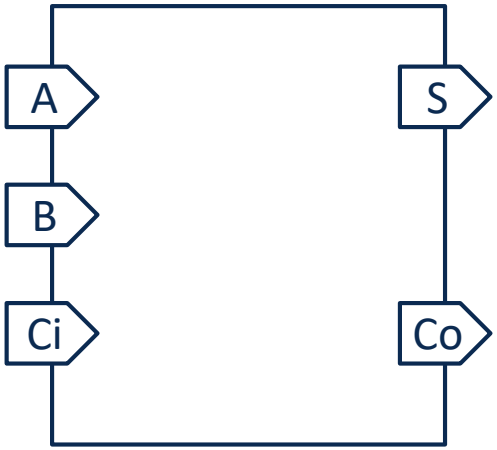
VHDL Design Units

- **entity**
  port declaration
- **architecture**
  internal functionality of a design unit
- **configuration**
  select one of several architectures

- **package**
  commonly used declarations, such as data types or functions
- **package body**
  implementation of a package

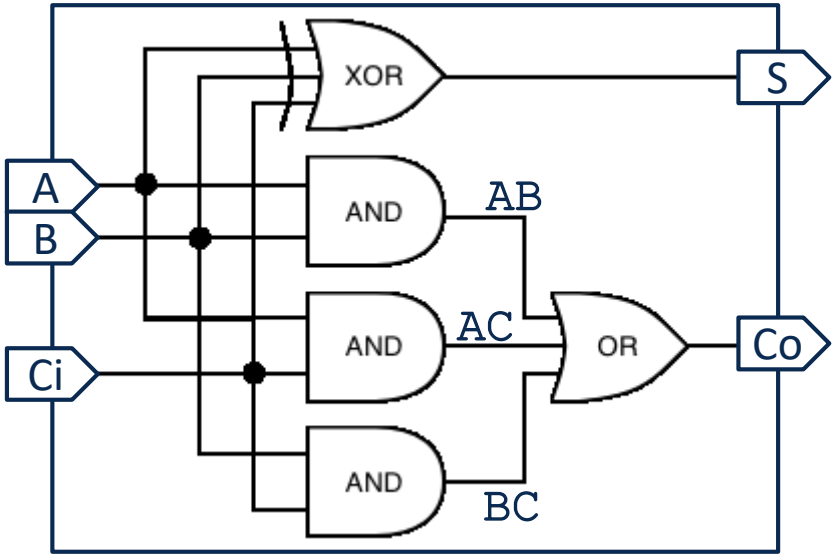
entity – defines the input and output **ports** of a design unit

```
entity FULL_ADDER is
   port( A : in std_logic;
         B : in std_logic;
         Ci: in std_logic;
         S : out std_logic;
         Co: out std_logic);
end entity FULL_ADDER;
```



architecture – describes **functionality** and **internal structure** of a design unit

```
architecture RTL of FULL_ADDER is
   signal AB, AC, BC: std_logic;
begin
   S  <= A xor B xor Ci;
   AB <= A and B;
   AC <= A and Ci;
   BC <= B and Ci;
   Co <= AB or AC or BC;
end architecture RTL;
```

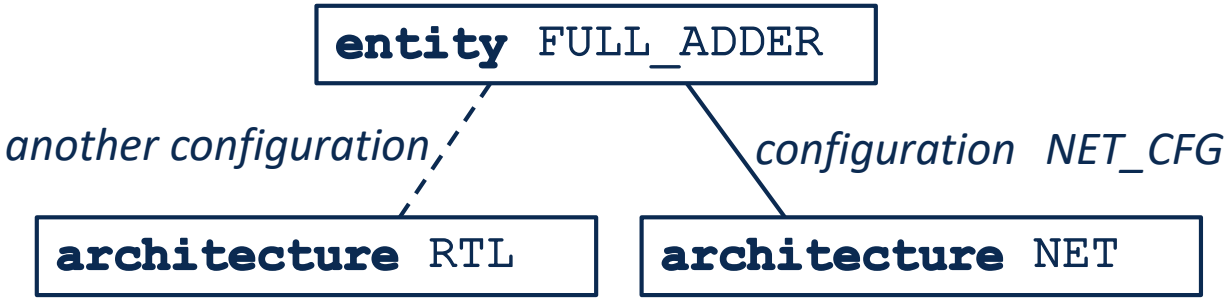


another **architecture** of the same entity

```
architecture NET of FULL_ADDER is
   -- component declaration – we come to it later
begin
   XOR3_i  : XOR3 port map (I1 => A, I2 => B, I3 => Ci, O => S);
   AND2_i1: AND2 port map (I1 => A, I2 => B, O => AB);
   AND2_i2: AND2 port map (I1 => A, I2 => C, O => AC);
   AND2_i3: AND2 port map (I1 => B, I2 => C, O => BC);
   OR3_i   : OR3  port map (I1 => AB, I2 => AC, I3 => BC,
                            O => Co);
end architecture NET;
```

configuration – *optionally* select particular architectures

```
configuration NET_CFG of FULL_ADDER is
   for NET
      -- select instance configurations – we come to it later
   end for
end NET_CFG;
```



Use Model for entity / architecture / configuration

- very often only one architecture
  - ➢ entity and architecture in one file and no configuration
- use configuration, when default binding is not sufficient

```
entity FULL_ADDER is
   port(...);
end entity FULL_ADDER;

architecture RTL of FULL_ADDER is
   ...
end architecture RTL;
```

VHDL entity

```
entity FULL_ADDER is

   port( A : in std_logic;
         B : in std_logic;
         Ci: in std_logic;
         S : out std_logic;
         Co: out std_logic );

end entity FULL_ADDER;
```

name of the design unit

port direction: in | out | inout

port data type

port name

no semicolon after last element

closing parenthesis of the port statement

VHDL architecture

name of the architecture
use telling names: RTL, BEH, NET

name of the design unit

```
architecture RTL of FULL_ADDER is
   signal AB, AC, BC: std_logic;
begin
   S  <= A xor B xor Ci;
   AB <= A and B;
   AC <= A and Ci;
   BC <= B and Ci;
   Co <= AB or AC or BC;
end architecture RTL;
```

internal signals
correspond to
physical connections
in the real world

actual data processing
in parallel statements

Some Types of Parallel Statements

```
S  <= A xor B xor Ci;
```

signal assignment

```
process (A, B) is
begin
   AB <= A and B;
end process;
```

process statement

```
OR3_i  : OR3
     port map (I1 => AB, I2 => AC, I3 => BC, O => Co);
```

instantiation statement

```
S  <= A xor B xor Ci;


process (A, B) is
begin
   AB <= A and B;
end process;


OR3_i  : OR3
     port map (I1 => AB, I2 => AC, I3 => BC, O =>
Co);
```

before we can explain statements and expressions, we need some **Data Types**

VHDL Data Types

- **Pre-defined Data Types**

`integer`	arbitrary range, defaults to 32bit
`natural`	non-negative integers e.g. indices into arrays and vectors
`real`	digital models of analog behaviour
`time`	advance time in test benches model timing behavior
`boolean`	truth values e.g. results of comparisons
`bit, bit_vector`	built-in logic type, ***don't use them***

VHDL Data Types

- **User-defined types**

  enumerations
  Just a list of names. Very good for FSMs, commands, etc.

  records
  Aggregate of arbitrary types.
  Good for behavioral models and test benches.

- **IEEE 1164 logic** is the most prominent enumeration type

`std_logic`	single-bit digital signals
`std_logic_vector`	multi-bit digital busses
`signed, unsigned`	`std_logic_vector` in numerics

=> preferred type to design digital circuits

9 Values of `std_logic`

'U' – **uninitialized** before anything is assigned to a signal

'X' – **unknown** unresolved or conflicting condition

'0' – **logic 0**

'1' – **logic 1**

'Z' – **high impedance** tri-state signal, undriven

'W' – **weak** conflicting condition or otherwise unknown state, can be over-written by '1' and '0'

'L' – **weak 0** weakly driven '0', e.g. pull-down

'H' – **weak 1** weakly driven '1', e.g. pull-up

'–' – **don't care** useful in selectors

9 Values of `std_logic`

- **'U'** – **uninitialized** before anything is assigned to a signal
- **'X'** – **unknown** unresolvable or conflicting condition
- **'0'** – **logic 0**
- **'1'** – **logic 1**
- **'Z'** – **high impedance** tri-state s[illegible], undriven
- **'W'** – **weak** conflicting conditi[illegible] or otherwise unknown state, can be over-written b[illegible] '1' and '0'
- **'L'** – **weak 0** weakly d[illegible]ven '0', e.g. pull-down
- **'H'** – **weak 1** weakly driven '1', e.g. pull-up
- **'–'** – **don't care** useful in selectors

use these for digital design

9 Values of `std_logic`

- **'U'** – **uninitialized** before anything is assigned to a signal
- **'X'** – **unknown** unresolvable or conflicting condition
- **'0'** – **logic 0**
- **'1'** – **logic 1**
- **'Z'** – **high impedance** tri-state, undriven
- **'W'** – **weak** conflicting condition or otherwise unknown state, can be over-written by '1' and '0'
- **'L'** – **weak 0** weakly driven '0', e.g. pull-down
- **'H'** – **weak 1** weakly driven '1', e.g. pull-up
- **'–'** – **don't care** useful in selectors

add those for modelling

Vectors

single bit signal

single bit data type

```
signal A : std_logic;
signal B : std_logic_vector(9 downto 0);
```

multi-bit bus

array data type

index range

```
0 to 9       -- increasing
9 downto 0   -- decreasing
```

Access to Vector Elements

```
architecture RTL of EXAMPLE is
   signal A : std_logic;
   signal B : std_logic_vector(9 downto 0);
   signal C : std_logic_vector(3 downto 0);
   signal D : std_logic_vector(5 downto 0);
begin
   A <= B(9);
   C <= B(8 downto 5);
   D(2 downto 0) <= B(4 downto 2);
end
```

single bit access

slice of 4 bits

slice assigned to

Define a New Array Type

name of the new array type

single-item base type

```
type something_vector is
 array (natural range <>) of something;
```

index definition:
a range of non-negative numbers

empty range:
to be filled in, when declaring objects of the new type

```
signal my4things : something_vector(3 downto 0);
```

Arrays of Arrays

```
type ram_type is array (0 to 255) of
            std_logic_vector(7 downto 0);
signal ram : ram_type;
```

data object of the new type

array type used for the single item type

Access to elements of arrays of arrays

```
type ram_type is
    array (0 to 511) of std_logic_vector(7 downto 0);
signal ram : ram_type;
signal read_data : std_logic_vector(7 downto 0);
signal address : std_logic_vector(9 downto 0);
signal some_bit : std_logic;
signal bit_select : std_logic_vector(2 downto 0);

read_data <= ram( to_integer(unsigned( address)) );
some_bit  <=
       read_data( to_integer(unsigned( bit_select)) );
```

Access to elements of arrays of arrays

```
type ram_type is
    array (0 to 511) of std_logic_vector(7 downto 0);
signal ram : ram_type;
signal read_data : std_logic_vector(7 downto 0);
signal address : std_logic_vector(9 downto 0);
signal some_bit : std_logic;
signal bit_select : std_logic_vector(2 downto 0);

read_data <= ram( to_integer(unsigned( address)) );
some_bit  <=
       read_data( to_integer(unsigned( bit_select)) );
```

use a signal of the single-element type

Caveat: Longest Static Prefix

```
architecture behave of mug is
    signal sig : std_logic_vector(7 downto 0);
begin

process (sig)
    begin
        for i in 0 to 2 loop
            sig(i) <= '0';
        end loop;
end process;

sig(3) <= '1';

end behave;
```

This defines drivers for *ALL* bits of sig ...

... regardless of the actual values of *i*.

Therefore, sig(3) is reassigned to `U´

What is the v of sig after 10 ns?

A: 00001000

B: UUUU1000

C: UUUUU000

D: 11111000

Caveat: Longest Static Prefix